DOI: 10.1111/mice.12449

ORIGINAL ARTICLE

COMPUTER-AIDED CIVIL AND INFRASTRUCTURE ENGINEERING WILEY

Pruning deep convolutional neural networks for efficient edge computing in condition assessment of infrastructures

Rih-Teng Wu[1] | **Ankush Singla**[2] | **Mohammad R. Jahanshahi**[1,3] | **Elisa Bertino**[2] | **Bong Jun Ko**[4] | **Dinesh Verma**[4]

[1]Lyles School of Civil Engineering, Purdue University, West Lafayette, Indiana

[2]Department of Computer Science, Purdue University, West Lafayette, Indiana

[3]School of Electrical and Computer Engineering, Purdue University, West Lafayette, Indiana

[4]IBM Thomas J. Watson Research Center, Yorktown Heights, New York

Correspondence
Mohammad R. Jahanshahi, Lyles School of Civil Engineering, School of Electrical and Computer Engineering, Purdue University, West Lafayette, IN.
Email: jahansha@purdue.edu

Funding information
National Science Foundation, Grant/Award Number: IIS-1636891

Abstract

Health monitoring of civil infrastructures is a key application of Internet of things (IoT), while edge computing is an important component of IoT. In this context, swarms of autonomous inspection robots, which can replace current manual inspections, are examples of edge devices. Incorporation of pretrained deep learning algorithms into these robots for autonomous damage detection is a challenging problem since these devices are typically limited in computing and memory resources. This study introduces a solution based on network pruning using Taylor expansion to utilize pretrained deep convolutional neural networks for efficient edge computing and incorporation into inspection robots. Results from comprehensive experiments on two pretrained networks (i.e., VGG16 and ResNet18) and two types of prevalent surface defects (i.e., crack and corrosion) are presented and discussed in detail with respect to performance, memory demands, and the inference time for damage detection. It is shown that the proposed approach significantly enhances resource efficiency without decreasing damage detection performance.

1 | INTRODUCTION

1.1 | Background and motivation

According to the 2017 American Society of Civil Engineers (ASCE) infrastructure report (ASCE, 2017), the estimated cost for infrastructure rehabilitation, such as bridges, dams, and levees, requires billions of dollars. Early detection of deficiencies in structural components and surface defects can help in reducing the retrofit costs (Amezquita-Sanchez & Adeli, 2015; Choi, Yeum, Dyke, & Jahanshahi, 2018; Kong & Li, 2018; Li, Park, & Adeli, 2017; Oh, Kim, Kim, Park, & Adeli, 2017; S. W. Park, Park, Kim, & Adeli, 2015; K. Park, Torbol, & Kim, 2018; Qarib & Adeli, 2014; Rafiei & Adeli, 2017, 2018; Rafiei, Khushefati, Demirboga, & Adeli, 2017; Wu & Jahanshahi, 2018a), but current practice in structural health monitoring (SHM) still requires manual inspection that is labor-intensive and time-consuming. The development of cost-effective and autonomous SHM approaches is an urgent need in order to enhance the efficiency of inspection processes (Bertino & Jahanshahi, 2018).

Recent advances in sensor technology and artificial intelligence provide opportunities for developing novel SHM approaches. The employment of sensor networks along with the communication capabilities among sensors has established the notion of Internet of things (IoT) (Tang, Sun, Liu, & Gaudiot, 2017). The IoT consists of a set of edge sensors and central server units. In an IoT system, decision-making can take place at the edge or at a server, depending on the application's requirements (Shi, Cao, Zhang, Li, & Xu, 2016; Verma & Mel, 2017). For civil infrastructure, the inspection target is usually enormous in size or length. A decision-making scheme where all the data collected at the edge sensor are transmitted to a server and the decisions

made on the server are sent back to the edge device would be extremely inefficient, particularly because the transmission bandwidth is often limited. A preferable solution, which is referred to as edge computing (Shi et al., 2016), is to deploy edge devices that have the capability to analyze data, and make decisions about data acquisition without the support of a remote server. For instance, a self-driving vehicle is equipped with sensors such as GPS, the ultrasonic sensor, the camera, and the light detection and ranging (LiDAR) sensor. The information acquired from the sensors enables the vehicle to make real-time decisions without central control unit. For the foreseeable future, with the development of swarm robotics (Brambilla, Ferrante, Birattari, & Dorigo, 2013) being mature, an autonomous inspection system will leverage the coordination between robots. Each individual robot, as an edge device, can make its own decision and communicate with other robots to find any potential presence of damage efficiently. In this way, the bandwidth and time spent on the data transmission are saved and more rapid inspections are achieved. However, such an approach requires the optimization in memory and computing costs of SHM algorithms deployed at the edge devices as these devices may have limited memory and computational resources.

Deep convolutional neural networks (DCNNs), due to their ability to automatically extract features, have been successfully used in computer vision since the breakthrough of the 2012 ImageNet challenge (Krizhevsky, Sutskever, & Hinton, 2012). The convolution kernels in DCNN capture the spatial invariant characteristics such as edges and contrast from the input image where these features are then used to make inference about the image. Since 2017, the rapid growth of DCNN-based approaches for damage detection in civil engineering has shown huge potential (Atha & Jahanshahi, 2018; Cha, Choi, & Büyüköztürk, 2017; Cha, Choi, Suh, Mahmoudkhani, & Büyüköztürk, 2018; Chen & Jahanshahi, 2018; Gao & Mosalam, 2018; Kumar, Abraham, Jahanshahi, Iseley, & Starr, 2018; Lin, Nie, & Ma, 2017; Yeum, Dyke, Ramirez, & Benes, 2016; Yeum, Dyke, & Ramirez, 2018; Wu & Jahanshahi, 2018b; Xue & Li, 2018; Zhang et al., 2017). However, the high computation and memory demands required for DCNN make it inappropriate for deployment on mobile inspection devices, such as unmanned aerial vehicles (UAVs) and robots. Furthermore, a DCNN needs a large volume of training data, which is sometimes not feasible for a newly discovered damage pattern, to avoid network overfitting. To address these issues, the concepts of transfer learning and network pruning have been introduced (Atha & Jahanshahi, 2018; Gao & Mosalam, 2018; Han, Pool, Tran, & Dally, 2015; Molchanov, Tyree, Karras, Aila, & Kautz, 2017).

In general, there are four schemes to perform image-based damage detection using machine learning techniques. *Scheme 1*: Extract handcrafted features, for example, local binary patterns (Ojala, Pietikainen, & Maenpaa, 2002) and histogram of gradients (Dalal & Triggs, 2005), from the input images. Use these features to train a classifier, for example, support vector machine (SVM) (Chen, Jahanshahi, Wu, & Joffe, 2017). *Scheme 2*: Design a DCNN from scratch and train it using sufficient labeled training data, for example, the NB-CNN network by Chen and Jahanshahi (2018). *Scheme 3*: Start from a pretrained network, for example, VGG16 (Simonyan & Zisserman, 2014), replace the fully connected (FC) layers with new FC layers or other classifiers like SVM and k-nearest neighbor (KNN). The pretrained network serves as an autonomous feature extractor. *Scheme 4*: Start from a pretrained network, fine-tune the network, and reduce its size by pruning technique (Molchanov et al., 2017). The first scheme is able to classify the images with low memory demands and fast inference time, but it may fail to achieve good performance when the images are collected under various environmental conditions, for example, illumination, due to the use of engineered features. Depending on the size of the designed DCNN, scheme 2 could be deployed onto computing platforms with small memory and low computing power. However, it is necessary to have enough labeled data for training. To address the problem of insufficient training data, scheme 3, referred to as *transfer learning*, can be adapted to a new classification problem with a small amount of training data by using a pretrained network. However, pretrained networks are usually quite large in size, and hence suffer from high memory and computation costs. The proposed scheme 4 exploits the advantages of transfer learning, and enhances the efficiency in memory cost and inference time for field applications by the Taylor expansion-based network pruning technique (Molchanov et al., 2017).

1.2 | Related work in network pruning

Recently, there have been some efforts in the area of network pruning. Han et al. (2015) removed the filters if their values were below a predefined threshold after fine-tuning with a strong regularization term. This approach requires a long time for network fine-tuning. Guo, Yao, and Chen (2016) proposed an approach called dynamic surgery network to evaluate the connections between neurons where the network structure was continuously maintained during the process of determining the least important connections. Han, Mao, & Dally (2016) proposed a three-stage pruning pipeline: pruning, K-means quantization, and Huffman coding. The first stage removed the unimportant connections between neurons, while the last two stages reduced the memory demands of the network by the quantization of the weight values and Huffman encoding for lossless data compression. Aghasi, Abdi, Nguyen, and Romberg (2017) implemented network pruning by solving a convex optimization problem, in which the sparsest set of weights for each layer were identified and then set to zero.

In A. Zhou, Yao, Guo, Xu, and Chen (2017) an incremental network quantization approach was proposed to reduce the memory storage. Instead of using the full-precision (i.e., 32-bit floating-point), the proposed approach converted the weight values to be either powers of two or zero to enhance the storage efficiency. Yu et al. (2017) proposed a network pruning approach based on the capability of parallelism of the computing units. Luo, Wu, and Lin (2017) evaluated whether a filter can be removed or not based on the outputs of its next layer rather than its own layer.

Mallya and Lazebnik (2018) proposed an iterative pruning approach to perform multitasks classification from a single network. Starting with a network trained for one task, the neurons with weights of smaller absolute magnitude were set to zeros, and these neurons were retrained for another classification task. Following this iterative pruning and retraining process, the resulting network was able to deal with multitasks at the same time. Liu, Sun, Zhou, Huang, and Darrell (2018) evaluated the existing pruning algorithms in terms of classification accuracy. Extensive experiments showed that for pruning algorithms that use a predefined target network architecture, similar performance can be achieved by training the target architecture from scratch. For pruning algorithms that automatically determine the target architecture by the evaluation of the global importance of the filters, the value of these algorithms lies in the searching of efficient architectures or sparsity patterns. In Hu, Peng, Tai, and Tang (2018), the filters with high percentage of zero activations were removed from the network by evaluating the network on the validation dataset. Most recently, He et al. (2019) proposed a reinforcement learning-based pruning algorithm to reduce the computation costs of deep neural networks. The proposed algorithm removed the filters through the rewards obtained from the predictions of the network. It is noted that some of the aforementioned approaches were relatively naive and none of the them have been focusing on efficient edge computing for field applications related to civil infrastructure condition assessment.

In this work, the VGG16 (Simonyan & Zisserman, 2014) network is first analyzed to demonstrate the efficiency of inference before and after network pruning. Two types of surface defects, that is, crack and corrosion, are considered to show the effectiveness of the proposed approach for damage detection. NVIDIA TITAN X GPU and NVIDIA Jetson TX2 GPU development kit are selected as the computing platforms representing the server and the edge device, respectively. The ResNet18 (He, Zhang, Ren, & Sun, 2015a) network, which has a size smaller than VGG16, is then used for comparison with VGG16 in terms of damage detection performance, memory demands, and inference time. Moreover, an optimization method is proposed to further reduce the inference time of VGG16 by enhancing the efficiency in feature computation.

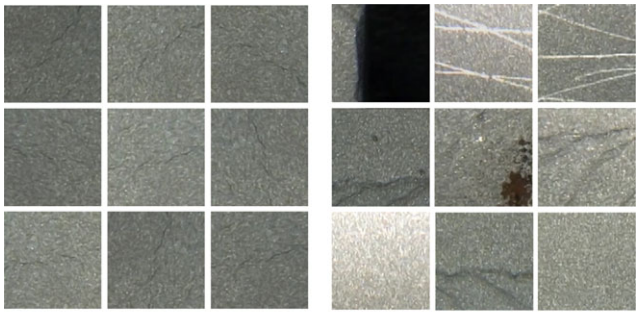


FIGURE 1 Crack dataset samples: (a) crack and (b) noncrack samples

1.3 | Contribution and scope

The contribution of this work is as follows. (a) Efficient DCNNs are designed and developed by using transfer learning and network pruning. (b) Experimental results on crack and corrosion detection with different computing platforms are presented to show the efficiency of the proposed DCNNs with respect to memory cost and inference time. (c) Two pretrained networks, that is, VGG16 and ResNet18, are experimentally tested for damage detection to show the effect of different pretrained networks on detection performance, memory demands, and inference time. (d) An optimized feature extraction approach is proposed to further reduce the inference time of VGG16.

The rest of the paper is organized as follows. Section 2 describes the image datasets and computing platforms used in this work. Section 3 elaborates on the concept and formulation of the network pruning technique as well as the proposed optimization method for the VGG16 feature extraction. The results and discussions about the conventional transfer learning approaches and the proposed efficient DCNN are provided in Section 4. Section 5 outlines conclusions and final remarks.

2 | DATASETS AND COMPUTING PLATFORMS

There are two types of common surface defects investigated in this work, that is, the crack and corrosion damage. The crack images are collected from the underwater inspection videos of internal nuclear power plant components. The image pixel resolution is 720×540, and a total of 147,344 and 149,460 crack and noncrack image patches with size 120×120 pixels are cropped from the original images (Chen & Jahanshahi, 2018). Figure 1 shows the crack and noncrack samples of the dataset. Due to the nature of metallic surfaces, the noncrack samples contain welds and grind marks, which makes it hard to differentiate them from the crack samples even for human inspectors. Implementing the proposed approach

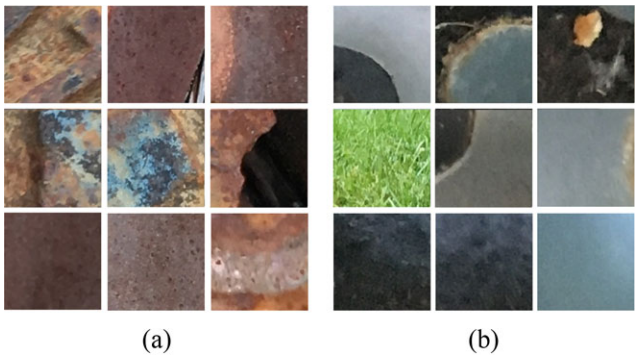


FIGURE 2 Corrosion dataset samples: (a) corrosion and (b) noncorrosion samples

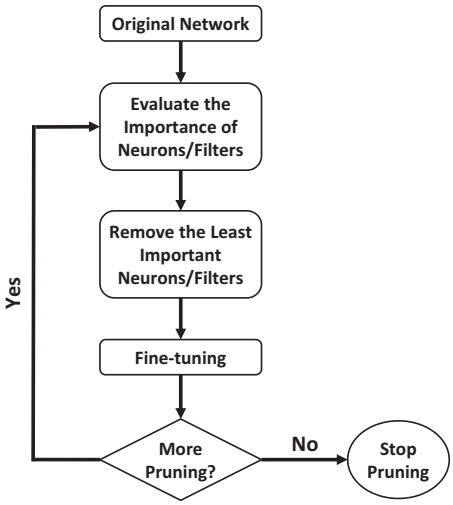


FIGURE 3 Network pruning flowchart

(i.e., Scheme 4) requires significantly less training data than a scheme that would train a DCNN from scratch (i.e., Scheme 2). To this end, only 29,468 crack (training: 25,048, testing: 4,420) and 29,780 noncrack (training: 25,313, testing: 4,467) image patches are used in this study. All the image patches are scaled to size 224×224 pixels for the pretrained networks.

Figure 2 shows corrosion and noncorrosion samples from the corrosion dataset collected on metallic surfaces. The image patches with size of 128×128 pixels are cropped from a total of 926 images captured using different digital cameras (Atha & Jahanshahi, 2018). This dataset contains a total of 33,039 corrosion (training: 28,083, testing: 4,956) and 34,148 noncorrosion (training: 29,026, testing: 5,122) image patches. Note that the underlying characteristics of the two datasets used in this study are completely different. A crack damage is usually thin in width, long in length, and darker than the background image. The crack dataset belongs to metallic submerged under water. Crack detection on metallic surfaces is quite challenging since the cracks are tiny and there is less contrast between the crack and its background compared to concrete surfaces. Furthermore, the existence of scratches, welds, grind marks, and varying light condition as well as light reflection make the crack detection task more challenging. The corrosion dataset consists of online images as well as images that are captured by using different cameras by the research team. A corrosion damage often covers a large area with contrast in color and texture. The use of datasets varying in nature demonstrates that the proposed approach is quite flexible.

Two different computing setups are used for experiments. The first is a server setup with high computing power and the second is an embedded chipset to represent the reduced computation capacity of an edge device. The experimental server setup includes an Intel Xeon processor E52620, 2.1 GHz with 16 GB RAM and an NVIDIA TITAN X GPU with 3584 CUDA cores at a base clock rate of 1,417 MHz and 12 GB GDDR5X memory. For the embedded chipset, the NVIDIA Jetson TX2 GPU is chosen as the platform since NVIDIA has demonstrated its capability of performing integrated operation for GPU computation and navigation (NVIDIA Jetson Solutions for Drones and UAVs, 2016; Akbar, Qidwai, & Jahanshahi, 2019). The Jetson TX2 is a developer kit equipped with a GPU with 256 CUDA cores and a combined 8GB LPDDR4 RAM along with a CPU with dual-core NVIDIA Denver2 + quad-core ARM Cortex-A57. Note that the main concerns are the inference time performance and the accuracy, not the training time, of the DCNN models.

3 | METHODOLOGY

Although the proposed approach is validated through the detection of crack and corrosion damage, it can be applied to any vision-based decision-making problem. As usually numerous types of damage exist (e.g., pavement pothole, concrete spalling, and exposed rebars), it is often very difficult to acquire sufficient data for network training at the beginning. Whenever a DCNN needs to be used to deal with a new damage detection problem from images, transfer learning is usually the best choice due to the limited available training data. However, as discussed in Section 1, such transfer learning-based methods are inappropriate when the decision-making processes are required to be made on edge devices. The use of network pruning is introduced to enhance the resource efficiency for onboard computations, and thus allows one to deploy DCNNs that are very accurate, have low storage and computing costs, and make decisions very quickly at the edge. Network pruning can be executed on the server machine, and the pruning terminates when either the detection performance on the test dataset starts to decrease or drops below a user-defined performance tolerance.

Figure 3 illustrates the pruning flowchart for constructing an efficient DCNN through transfer learning and

network pruning. The methodology starts with a pretrained network (e.g., VGG16), and then modifies its FC layers and retrains the network with a training dataset. As the network is originally designed for the ImageNet dataset consisting of 1,000 image categories, it is very large in size and may contain redundant convolution kernels that do not contribute to the new detection problems considered herein, that is, the detection of crack and corrosion. To reduce the network size, the importance of convolution kernels is evaluated through training data, and the kernels are removed based on their ranking in importance. Next, the pruned network is fine-tuned again to ensure its performance for damage detection. Based on the detection performance, the user can determine whether or not to further prune the network following the same procedure. Section 3.1 elaborates on the pruning technique that is used, and Section 3.2 describes the proposed optimization method for feature extraction in VGG16.

3.1 | Network pruning

Network pruning is closely related to the biological brain behavior. At the beginning, the brain learns to handle multitasks, and it becomes more efficient by disconnecting some of the neurons if it is asked to focus on a specific task. The concept of network pruning can be traced back to the work done by LeCun, Denker, Solla, Howard, and Jackel (1990), in which the generalization of a neural network is enhanced by the removal of the parameters selected using a second-order Taylor expansion.

To evaluate the parameters' contribution in the network, the existing techniques include, but are not limited to, the investigation of the statistical values of activations, the absolute value of kernel weights, the mutual information between the activations, and the network predictions (Han et al., 2015; Han, Liu et al., 2016; Hu, Peng, Tai, & Tang, 2016), regularization-based techniques (Alvarez & Salzmann, 2016; Lebedev & Lempitsky, 2016; H. Zhou, Alvarez, & Porikli, 2016), and the Taylor expansion-based approaches (Molchanov et al., 2017). In this work, the Taylor expansion-based algorithm is adopted for the following two reasons (Molchanov et al., 2017): (a) Compared to the regularization-based approaches, the Taylor expansion-based approach uses the global rescaling of criteria and therefore no computation on per-layer sensitivity analysis is needed. (b) The Taylor expansion-based algorithm is demonstrated to be more robust against other pruning methods in the sense that the algorithm keeps maintaining higher classification accuracy as the pruning proceeds.

Intuitively, the pruning algorithm aims to find the set of kernel weights W' that leads to the minimum changes in the cost function $C(\cdot)$:

$$\min |\Delta C(W)| = \left|C(D|W') - C(D|W)\right| \quad \text{s.t. } ||W'|| \leq B \tag{1}$$

where D denotes the training data and W denotes the set of the kernel weights obtained after network training is finished. B is the number of nonzero elements in W'. Notice that it requires $2^{|W|}$ evaluations on the kernel weights to solve Equation (1) if considering all the combinations among the elements in W. To reduce the computational costs for solving W', a greedy method is employed that iteratively searches among all the subsets of W' to be removed. Thus, such method removes subsets one at a time, as shown in Figure 3.

For DCNNs, the convolution feature maps can be considered as the elements of W'. Let h_i denote the ith feature map in a DCNN, $i = 1, \ldots, L$, where L is the total number of feature maps in the DCNN. By assuming independence of parameters, the cost function $C(D|W)$ can be written as:

$$C(D|W) \approx C(D|h_i) = C(D, h_i) \tag{2}$$

Let $C(D, h_i = 0)$ denote the cost function $C(\cdot)$ where the feature map h_i is being removed. Then, the difference in cost function is:

$$\left|\Delta C(h_i)\right| = \left|C(D, h_i = 0) - C(D, h_i)\right| \tag{3}$$

Using Taylor expansion, the term $C(D, h_i = 0)$ is approximated as

$$C(D, h_i = 0) = C(D, h_i) - \frac{\partial C}{\partial h_i} h_i + R_1(h_i = 0) \tag{4}$$

where $R_1(h_i = 0)$ is the first-order remainder, that is, the higher order terms. By ignoring $R_1(h_i = 0)$ and substituting Equation (4) into Equation (3), the change in the cost function is estimated as

$$\left|\Delta C(h_i)\right| = \left|\frac{\partial C}{\partial h_i} h_i\right| \tag{5}$$

Conceptually, the pruning algorithm removes the feature map h_i that leads to a near-zero gradient of the cost function. The difference in cost function can be computed through the standard backpropagation approach (Bottou, 2010). The pruning algorithm is implemented in Python 2 using PyTorch (Paszke et al., 2017) version 0.2 with CUDA 8.0, cuDNN 6.0.21, and Ubuntu 16.04.

3.1.1 | Pruning VGG16 network

VGG16 has been widely adopted for SHM applications in recent two years (Atha & Jahanshahi, 2018; Cao & Choe, 2018; Gopalakrishnan, Gholami, Vidyadharan, Choudhary, & Agrawal, 2018; Maeda, Sekimoto, Seto, Kashiyama, & Omata, 2018). As the winner of the 2014 ImageNet challenge, VGG16 is the first network that introduces the concept of deep neural networks. It has a total of 120,000,000 parameters, and the network architecture consists of simple concatenation of convolution and pooling layers (Simonyan & Zisserman, 2014).

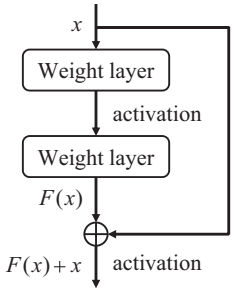


FIGURE 4 An illustration of a sample ResNet18 building block

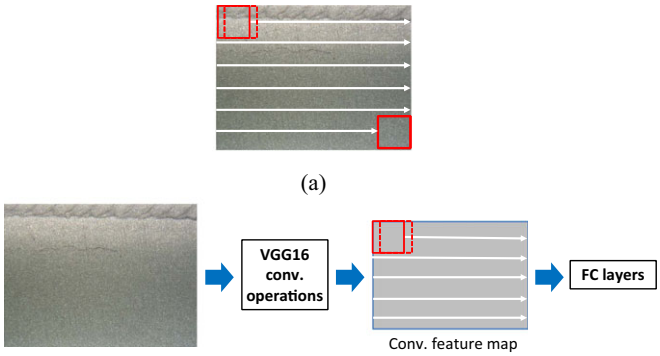


FIGURE 5 An optimization of feature extraction for VGG16: (a) pass each sliding window into VGG16 separately, and (b) pass the whole image frame into VGG16 to generate a convolutional feature map, and then extract the feature of each window from the corresponding location on the feature map

To use VGG16 for crack detection, the last FC layer in VGG16 is replaced with a binary output layer to indicate the presence of cracks in the input image patch. The stochastic gradient decent (SGD) algorithm (Bottou, 2010) with learning rate 0.001 and momentum 0.9 is used for network fine-tuning. The parameters of the whole original network are first fine-tuned with the crack training dataset, and the convolutional feature maps are removed based on the pruning steps illustrated in Figure 3. At each pruning iteration, the contributions of the convolutional kernels are ranked using training data, and 512 kernels are removed from a total of 4,224 convolutional kernels in VGG16. Next, the whole pruned network is again fine-tuned using the same learning rate with the training data to enhance its detection performance, and the user decides whether to further prune the network or not. In this work, the number of fine-tuning epochs for the original network is 20, and the number of fine-tuning epochs for the pruned network is 10. Seven pruning iterations are conducted for VGG16 to remove 84% of the convolution kernels. Following the same procedure, another DCNN is constructed for corrosion detection using the corrosion dataset.

3.1.2 | Pruning ResNet18 network

As the winner of the 2015 ImageNet challenge, ResNet18 introduces the concept of residual learning to make easier inference on the mapping between the input and output (He et al., 2015a). The network consists of the concatenations of residual learning blocks, as shown in Figure 4, to enhance network training through identity mapping. The total number of parameters in ResNet18 is 9,000,000, which is approximately 13 times less than VGG16.

To adopt ResNet18 for crack detection, the FC layer is replaced with a binary output layer. Using the same network fine-tuning and pruning procedure as VGG16, six pruning iterations are conducted for ResNet18 to remove 79% of the convolution kernels from a total of 3,904 kernels.

It is noted that the residual learning block requires the input and output dimensions to be identical, as shown in Figure 4. During the training phase, the output $F(x)$ of a residual learning block consists of weight layers and nonlinear activations are added to the input x. The resulting $F(x) + x$ is then passed to the subsequent layers for further processing. Since the convolution kernels are removed based on ranking of importance, the dimension compatibility of the input and output of the residual learning blocks is not guaranteed. For instance, given an input x of dimension $224 \times 224 \times 32$, the output $F(x)$ after two convolution layers with kernel sizes of $3 \times 3 \times 32 \times 32$ is still $224 \times 224 \times 32$ by zero-padding. After pruning, the two convolution layers could have kernels with different sizes, for example, $3 \times 3 \times 32 \times 32$ and $3 \times 3 \times 32 \times 16$, resulting in an output dimension of $224 \times 224 \times 16$ for $F(x)$. Therefore, the residual learning blocks are disabled during the fine-tuning of the pruned networks.

3.2 | An optimization for feature extraction in VGG16

To detect and locate the presence of damage given an input image frame, one common approach is to first scan the whole frame by sliding windows of the same size of the patch size used in network training. Next, each window is passed to the network for damage identification, and the damage in the input image is localized by grouping the overlapping/neighboring sliding windows identified as damage patches (Chen & Jahanshahi, 2018). The step size between adjacent windows is user-defined. For instance, a step size of 8 pixels between each sliding window is used by Chen and Jahanshahi (2018). A smaller step size will enhance the spatial resolution of the damage detection result, while the time required to process the whole image will increase due to higher number of image patches to be processed. Notice that the overlapping regions between adjacent sliding windows will be large if the chosen step size is small, which means that the two adjacent windows share a large number of features, as shown in Figure 5a. The time required to compute the

features can be reduced by eliminating the redundant computations in the overlapping regions.

To reduce the computation in processing the whole image, an optimization approach inspired by SPPNet (He, Zhang, Ren, & Sun, 2015b) is proposed to compute the VGG16 features. Figure 5b illustrates the proposed approach. Given an input image, the image is passed to the network for processing up to the layer before the FC layers. The resulting convolutional feature map is then used to extract the features of the sliding windows in the original image according to their corresponding locations in the map. For instance, a step size of 8 pixels between the sliding windows in the original image corresponds to a step size of 1 pixel in the convolutional feature map after three executions of pooling operations. Therefore, repetitive convolution operations are saved and the inference time for the whole frame greatly decreases. This can further improve the efficiency of DCNNs for deployment onto edge devices. Note that the proposed optimization approach can be applied to any network that consists of concatenations of convolution and pooling layers, for example, AlexNet (Krizhevsky et al., 2012). For networks with more complicated configuration, other optimization schemes may be required to eliminate the repetitive computation.

4 | RESULTS AND DISCUSSIONS

This section evaluates the damage detection performance, the inference time, and the memory requirement of the conventional transfer learning-based approaches (i.e., Scheme 3 in Section 1) compared to the proposed network pruning (i.e., Scheme 4 in Section 1) approach. The results on the crack and corrosion datasets are presented. In the experimental evaluation, the server setup and the edge device are used as computing setups with different computational capacities. During the training phase, only the server setup is used since all the heavy computations can be executed at a cloud server in practice. In the network testing phase, both the server setup and the edge device are used to compare the inference time and memory demands. The recorded inference time is the time required to classify 3,720 image patches with size 224×224, assuming that one 720×540 image has 3,720 sliding windows. Instead of using only 3,720 test image patches, the detection performance is based on the detection accuracy for all the image patches in the test datasets (i.e., 8,887 patches for crack and 10,078 patches for corrosion) to avoid any biased judgment.

4.1 | Conventional transfer learning without network pruning

Conventional transfer learning approaches use a pretrained network as a feature extractor, and train a new classifier based on these input features and the corresponding image labels. In this section, VGG16 (Simonyan & Zisserman, 2014) is adopted to extract $1 \times 4{,}096$ features given an input image patch of 224×224. The last two FC layers of VGG16 are replaced with a SVM or a KNN classifier. Three types of classifiers are considered, that is, KNN, the C-Support vector classifier (SVC), and the linear support vector classifier (SVMH). The Scikit-learn library (Pedregosa et al., 2011) is used, where the SVC classifier is implemented using LIBSVM (Chang & Lin, 2011) and SVMH classifier using LIBLINEAR (Fan, Chang, Hsieh, Wang, & Lin, 2008). The feature generation is implemented using TensorFlow library (Abadi et al., 2016) with Keras wrapper (Chollet, 2015).

Table 1 presents the experimental results for image-based damage assessment based on Scheme 3. The detection accuracy of the KNN classifier is slightly better than the SVC and the SVMH classifiers for the crack dataset, while the SVC classifier achieves better accuracy for the corrosion dataset. Among the three classifiers, the SVMH classifier has the least memory demand (i.e., 0.03 (MB)) and the smallest inference time (i.e., 234.8 and 245.7 s for crack and corrosion data, respectively) when deployed on the edge device. This means that it takes approximately 4 min to process one 720×540 image on the edge device that is inefficient for practical applications. For both crack and corrosion datasets, even with the server setup, it takes roughly 29 s to process one image for the SVMH classifier. Note that 99% of the inference time is spent on the feature extraction due to the heavy architecture of the network, which demonstrates the necessity of network pruning for efficient inference.

The feature extraction part is implemented using the TensorFlow Python library (Abadi et al., 2016), which utilizes the parallel processing capabilities of a GPU when available. The classifiers, on the other hand, use the Scikit-learn library (Pedregosa et al., 2011) that is a CPU-only implementation. Due to the powerful TITAN X GPU in the server setup, the feature extraction part is eight times faster on the server when compared to the edge device. However, the classifier part does not get similar speedup on the server since it utilizes only the CPU.

4.2 | Transfer learning with network pruning

4.2.1 | Pruning VGG16

To achieve efficient inference on the edge devices, the Taylor expansion-based network pruning technique described in Section 3.1 is used to reduce the size of the pretrained VGG16 network. Pruning is conducted on the server as it requires computations for ranking the convolution kernels. The pruned networks are then deployed to both the server and the edge device to test the inference time and the memory demands. Whenever the pruning algorithm removes 512 convolution kernels in each pruning iteration, the network is fine-tuned with 10 epochs. Figure 6 shows the

TABLE 1 Scheme 3 damage detection results: VGG16 used as feature extractor and KNN, SVC, and SVMH as classifier

Classifier	Data	Model size (MB)	Server setup inference time (s)	Edge device inference time (s)	Accuracy (%)
KNN	Crack	3,355	96.1	587.6	94.6
SVC	Crack	169	124.6	417.7	89.3
SVMH	Crack	0.03	29.5	234.8	85.5
KNN	Corrosion	3,787	89.0	548.4	82.8
SVC	Corrosion	191	131.7	492.5	89.8
SVMH	Corrosion	0.03	29.3	245.7	84.0

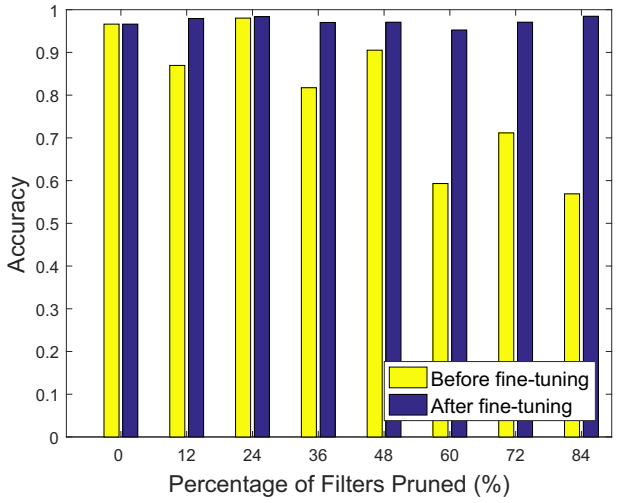


FIGURE 6 Detection performance of pruned VGG16 versus percentage of pruned filters: (a) accuracy of the 8,887 test image patches from the crack dataset and (b) accuracy of the 10,078 test image patches from the corrosion dataset. (Refer to Section 4.2.3 for more discussion about the stopping criterion for pruning and the effects of different fine-tuning datasets.)

detection performance before and after fine-tuning for both the crack and corrosion datasets. In general, the detection performance drops immediately when the number of convolution kernels is reduced, and then the accuracy is improved by fine-tuning with the training data. For the crack dataset, the accuracy before fine-tuning ranges from 56.9% to 98.0%, while the accuracy after fine-tuning lies in the range between 95.3% and 98.5%, for the pruned networks. It is noted that the detection accuracy after fine-tuning does not exhibit a monotonic decreasing behavior. This demonstrates that after pruning, the network still has the capacity to detect damage adequately since the original network is designed for ImageNet challenge to classify 1,000 categories. Also, a small oscillation is observed in the accuracy after fine-tuning because the number of fine-tuning epochs is fixed, and the experiment is conducted under the assumption that there is no sufficient training data. With enough fine-tuning and training data, the oscillation in detection accuracy should be reduced. For corrosion dataset, the accuracy before fine-tuning ranges from 79.4% to 90.0%, while the accuracy after fine-tuning lies in the range between 90.7% and 93.6%, for the pruned networks. This indicates that network fine-tuning is crucial to ensure good detection performance after pruning convolution kernels.

Figure 7 depicts the original VGG16 convolution feature maps versus the pruned network where 84% of convolution kernels are removed for crack detection. The numbers of the removed kernels in each convolution layer are: {1: 43; 2: 47; 3: 86; 4: 89; 5: 195; 6: 200; 7: 200; 8: 441; 9: 452; 10: 460; 11: 459; 12: 465; 13: 447}, where i: j denotes that j convolution kernels are removed in the ith convolution layer. As shown in Figure 7, after pruning, the network size is significantly reduced, and the memory demands of VGG16 drops from 525 to 125 MB, which is approximately an 80% reduction in memory. Figure 8 presents the distribution of the percentage of the pruned kernels in each convolution layer for both the crack and corrosion data using VGG16 network. According to the figure, the following observations are made: (a) The distribution for the corrosion data is close to the crack data. (b) A similar number of pruned kernels are deleted in the convolution layers between pooling layers. (c) Network pruning

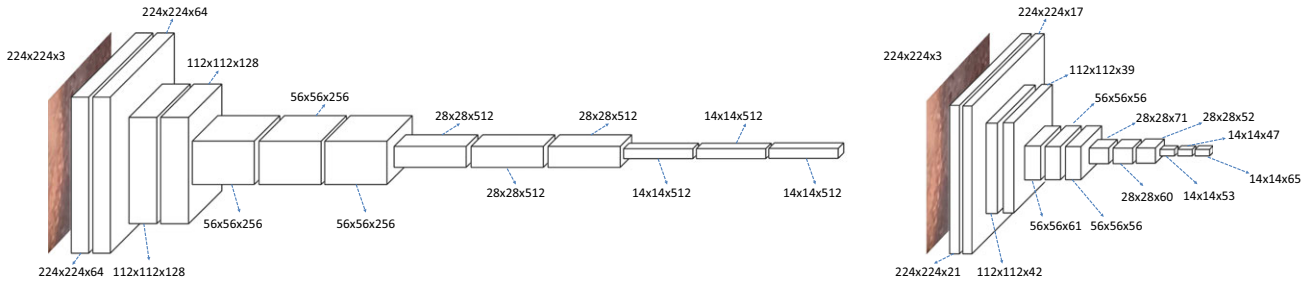


FIGURE 7 VGG16 convolution feature map dimensions: (a) original dimension of feature maps and (b) reduced feature map dimension after pruning 84% of the convolution kernels

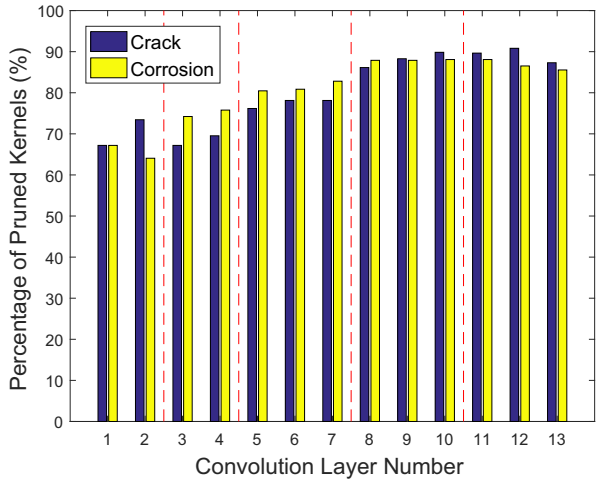


FIGURE 8 The distribution of the percentage of the pruned kernels in each convolution layer for the crack and corrosion datasets after 84% kernels being eliminated in the VGG16 network. The dashed lines indicate the locations of the pooling layers

removes more percentage of kernels in the deeper layer. This demonstrates that when using transfer learning for a new task very different from the original task, lower level feature representations extracted from the first few layers could be more useful than the higher level features in the deeper layer.

Figure 9 shows the variation of the inference time when using pruned networks for damage detection. For crack dataset, the inference time on edge device decreases from 279.7 to 31.6 s that corresponds to a reduction factor of 8.9. For the corrosion dataset, the inference time on edge device drops from 275.7 to 30.6 s that corresponds to a reduction factor of 9.0. This demonstrates how network pruning enhances the time efficiency on the edge. For the server setup, the inference time of crack and corrosion datasets decreases from 13.1 to 4.0 s and from 13.2 to 3.7 s, respectively. The corresponding reduction factors are 3.3 and 3.5, which is approximately 0.38 times of the reduction factors on the edge device. The main reason for this difference is the *cudaMalloc* function that allocates the memory on the GPU. The time taken by this function does not vary significantly when the size of the

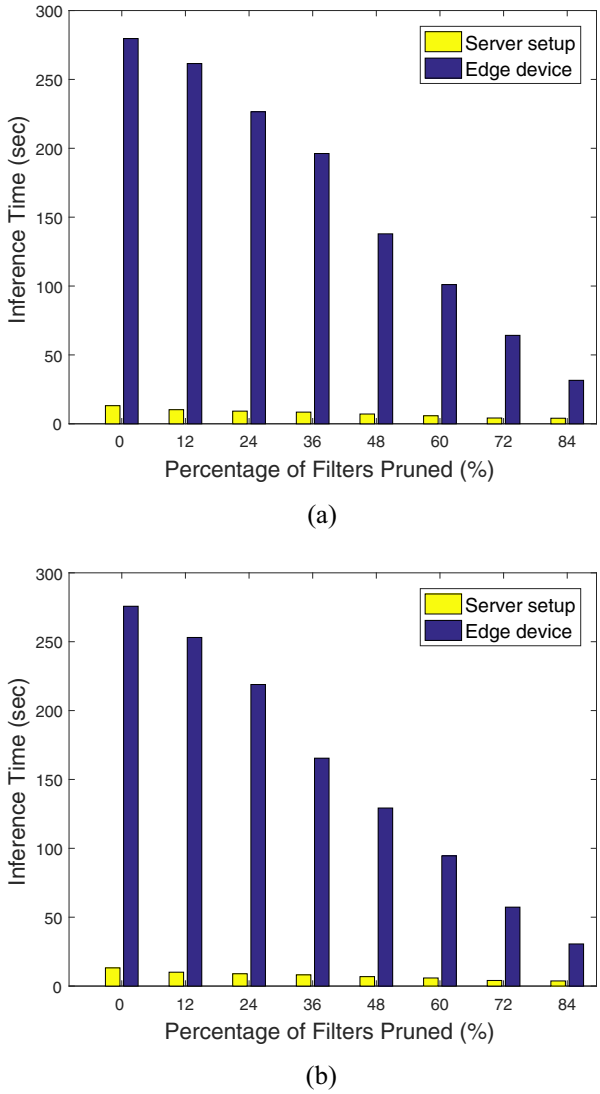


FIGURE 9 Inference time of VGG16 versus percentage of pruned filters: (a) crack dataset and (b) corrosion dataset. Inference time: the total time (in seconds) required for the forward-pass of 3,720 image patches of 224 × 224 pixels

model changes. For our implementation, *cudaMalloc* takes 2.3 s on the server setup, that is, approximately 60% of the total inference time in case of the pruned model, which makes the speedup less considerable. In case of the edge device, however, *cudaMalloc* takes 9.9 s which is only 33% of the inference time for a pruned network.

In addition, a CNN network (i.e., Scheme 2 in Section 1), which is a specialized network developed for crack detection by Chen and Jahanshahi (2018), is deployed on the server setup for comparison with the pruned VGG16 network. The specialized CNN takes 32.0 s to complete the inference, which is quite close to the time 31.6 s achieved by the VGG16 network with 84% kernels pruned. It is noted that the NB-CNN network uses a smaller input patch size of 120×120 pixels, meaning that the pruned VGG16 would be more efficient in inference time than the specialized CNN if using the same input size. This shows that network pruning provides a resource-efficient solution when using a large network with several layers and parameters at the beginning. However, the specialized CNN network is superior to the pruned VGG16 network in terms of memory demands and damage detection performance. The specialized CNN requires only 11.1 (MB) memory, and its detection accuracy is 99.5% due to the use of larger training data (i.e., 240,000 image patches).

It is shown that the pruned networks need fine-tuning with the training data to improve the detection performance. The more the fine-tuning is performed on a network, the longer it takes to complete the pruning on the server machine. Therefore, a sensitivity analysis is performed on the required number of fine-tuning epochs for both the crack and the corrosion datasets. Figure 10 shows the detection performance when fine-tuning the network with 1, 2, 5, and 10 epochs. As shown in the figure, there is no significant difference in the detection accuracy when using different numbers of fine-tuning epochs. For the crack dataset, the detection accuracy of the network with 84% kernels pruned is 94.9% and 98.5% when using 1 and 10 fine-tuning epochs, respectively. Similarly, for the corrosion dataset, the detection performance of the pruned network with 84% kernel removal is 92.2% and 90.7% when using 1 and 10 fine-tuning epochs, respectively. To eliminate the effects of random initialization, five repeat trials are conducted for each case of fine-tuning epochs when pruning 84% of the convolution kernels. The boxplots shown in Figure 11 are used to show the variation of the F-score in damage detection.

According to Figure 11, the variation in F-score is the smallest for both the crack and corrosion datasets when using five fine-tuning epochs. The median F-score for crack dataset is 0.963 and 0.981 for 1 and 10 fine-tuning epochs, respectively. The median F-score for corrosion dataset is 0.905 and 0.931 for 1 and 10 fine-tuning epochs, respectively. These results show that using 10 epochs to fine-tune the network

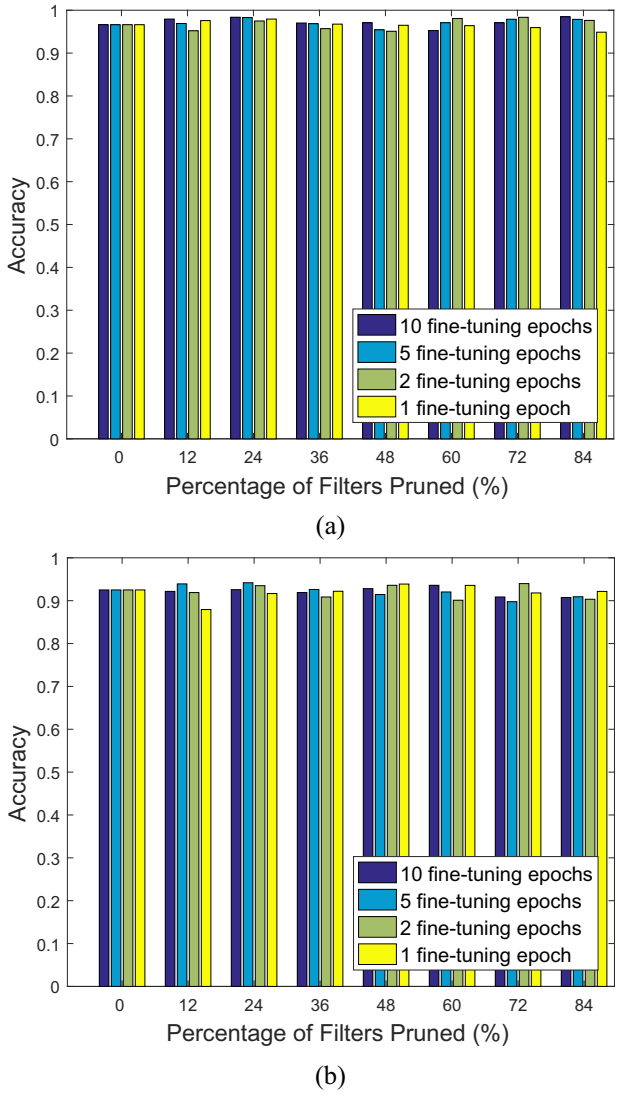
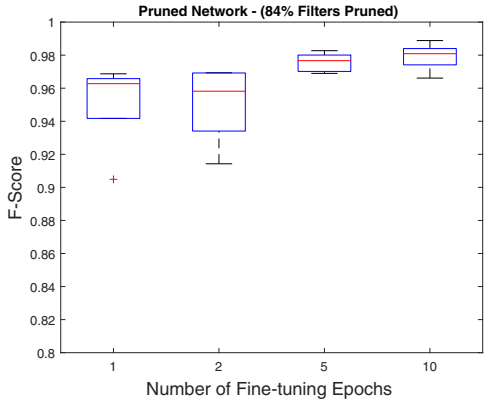


FIGURE 10 Effect of fine-tuning epochs on detection performance for VGG16 network: (a) accuracy of the crack test dataset and (b) accuracy of the corrosion test dataset

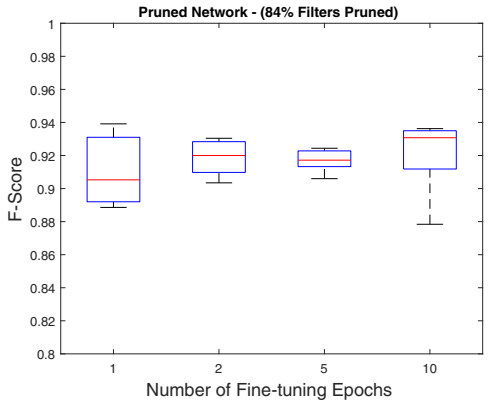
may achieve slightly better performance than using 1 epoch. Figure 12 illustrates the variation of time spent on network pruning. For the crack dataset, it takes a total of 1.55 and 7.16 hr to prune 84% of kernels using 1 and 10 fine-tuning epochs, respectively. For the corrosion dataset, it takes 1.65 and 7.75 hr to prune 84% of kernels with 1 and 10 fine-tune epochs, respectively. This demonstrates that using a smaller number of fine-tuning epochs reduces the time required for pruning on the server machine, and doing this does not lead to a significant decrease in the detection performance.

4.2.2 | Pruning ResNet18

Besides VGG16, the pretrained ResNet18 network is also adopted for detecting surface defects. The original ResNet18 network is 44 MB, which is 8.4% of the original VGG16 network size (525 MB) and therefore should be more efficient

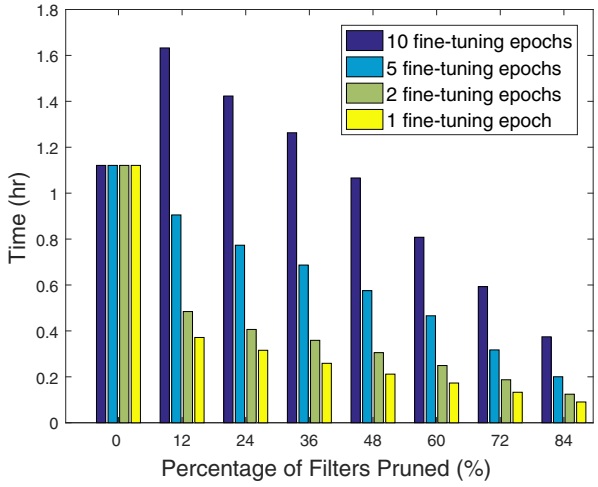


(a)

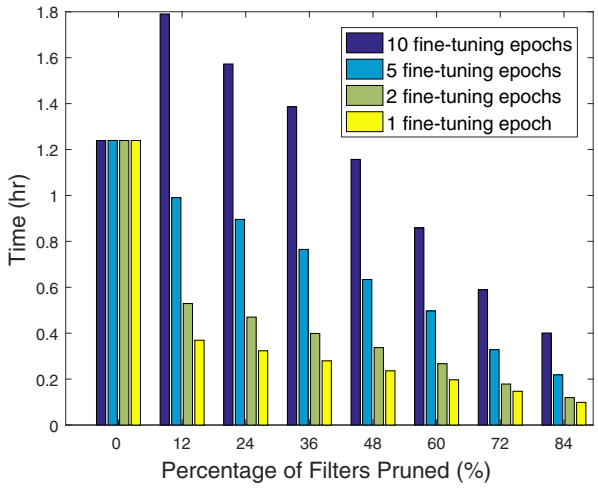


(b)

FIGURE 11 Repeated trials for damage detection with 1, 2, 5, and 10 fine-tuning epochs for pruned VGG16 network: (a) *F*-score of the detection results for the crack test data and (b) *F*-score of the detection results for the corrosion test data



(a)



(b)

FIGURE 12 Effect of fine-tuning epochs on pruning time of VGG16: (a) crack dataset and (b) corrosion dataset

with respect to inference time and memory costs. Figure 13 shows the detection performance of ResNet18 for the crack and corrosion datasets. Similar to VGG16, the detection accuracy decreases every time the algorithm removes 512 convolution kernels. For the crack dataset, the accuracy before fine-tuning is 49.7%, while for the corrosion dataset, the accuracy before fine-tuning ranges from 49.2% to 50.8%, for the pruned networks. After fine-tuning on the pruned networks, the accuracy improves and lies between 94.2% and 97.6% for the crack dataset. For the corrosion dataset, the detection accuracy increases to values between 91.8% and 93.1% after network fine-tuning. This again demonstrates that it is essential to fine-tune the network after network pruning. Figure 14 compares the inference time of VGG16 and ResNet18 when deployed on the edge device for damage detection. By removing 84% and 79% of the convolution kernels from VGG16 and ResNet18, the inference time for crack detection decreases from 279.7 to 31.6 s for VGG16 and from 36.8 to 8.9 s for ResNet18. For the corrosion dataset, the inference time decreases from 275.7 to 30.6 s and from 34.1 to 9.0 s for VGG16 and ResNet18, respectively. As expected, ResNet18 is more efficient with respect to the inference time (approximately 3.55 times faster than VGG16 at the end of network pruning). The ResNet18 size decreases from 44 to 2 MB, which is a 95% reduction compared to the original network size. Notice that VGG16 achieves an 89% reduction in inference time and 80% reduction in memory, while ResNet18 achieves a 76% reduction in inference time and 95% reduction in memory demands. This means that through network pruning, the gain for VGG16 is more prominent with respect to the inference time, while the gain for ResNet18 is more prominent with respect to memory requirement.

Although ResNet18 is more efficient than VGG16 in terms of inference time and memory costs, VGG16 has higher adaptability for new classification problems due to being a larger network. As shown in Figures 6 and 13, the detection performance of VGG16 drops approximately from 10% to 20% each time the network is pruned. For instance, in Figure 6a, the

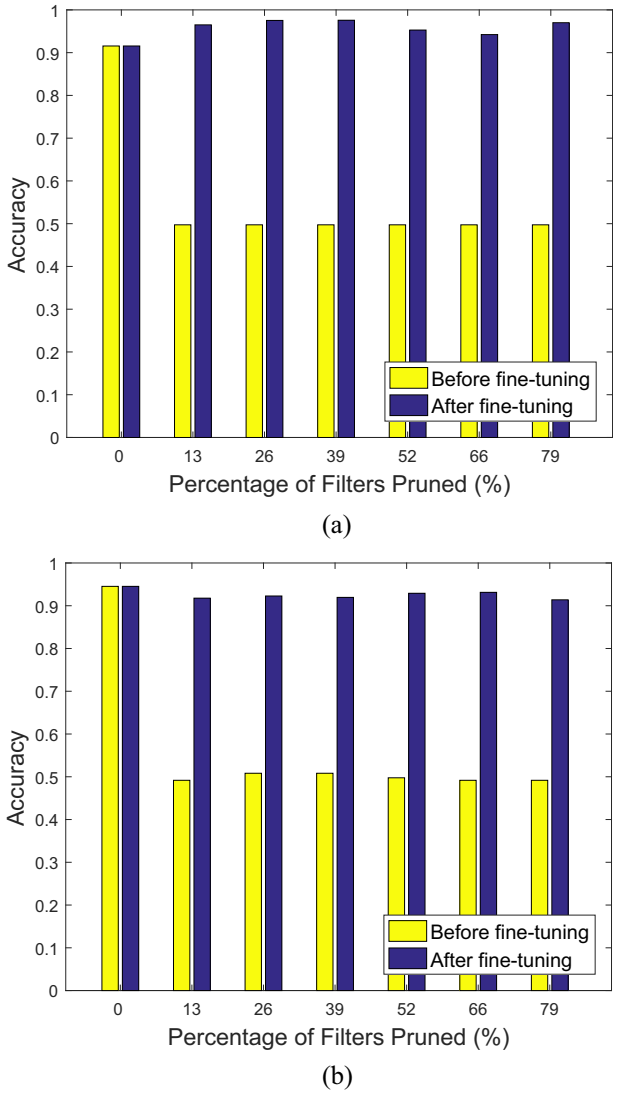


FIGURE 13 Detection performance of pruned ResNet18 versus percentage of pruned filters: (a) accuracy of 8,887 test image patches from the crack dataset and (b) accuracy of 10,078 test image patches from the corrosion dataset

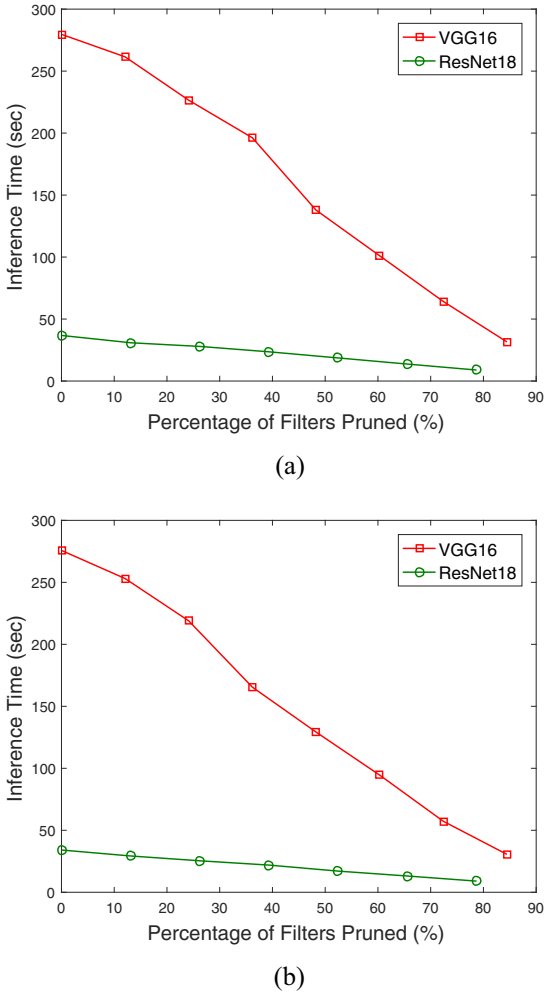
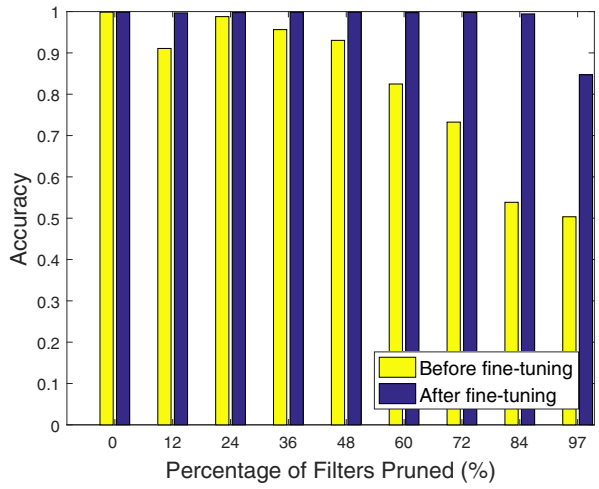


FIGURE 14 Inference time required for 3,720 image patches operated on Jetson TX2 GPU for VGG16 and ResNet18 networks: (a) crack dataset and (b) corrosion dataset

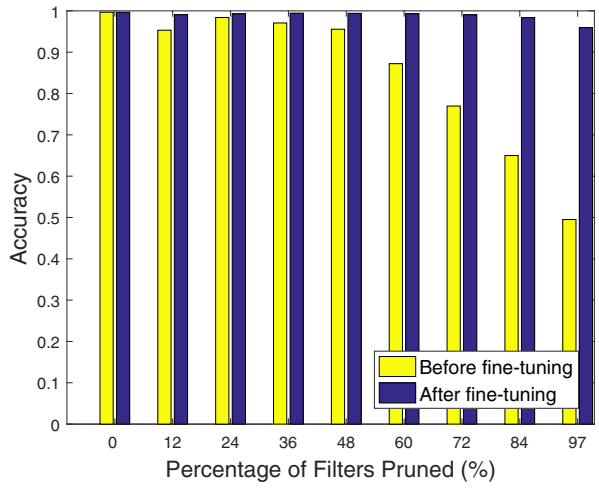
accuracy is 98.4% after fine-tuning when 24% of filters are removed, and the accuracy is 81.7% before fine-tuning when 36% of filters are removed. There is a 16.7% drop in accuracy because of pruning. Once the fine-tuning is applied, the accuracy reaches 97% although 36% of the filters are removed. However, the accuracy of ResNet18 drops significantly, that is, more than 30%, in each pruning iteration. Such result indicates that ResNet18 is more sensitive to changes in the network.

4.2.3 | Cross-validation and stopping criterion for pruning

As discussed in Section 3, network pruning may terminate when the detection performance on test datasets starts to decrease. In this section, pruning is conducted on VGG16 for both crack and corrosion datasets, and it is stopped if the detection accuracy after fine-tuning drops more than 3%. Fivefold cross-validation with the whole dataset is used to show the statistics of the detection performance, and therefore eliminates the bias induced by fine-tuning and testing on a particular training/testing dataset. In other words, five repeated trials are conducted for both crack and corrosion datasets. Each trial uses different training data, and the test datasets in the five trials are completely independent. Figure 15 reports the mean of the damage detection accuracy for both crack and corrosion data from fivefold cross-validation. Tables 2 and 3 report the mean (μ) and standard deviation (σ) of the accuracy before and after fine-tuning for crack and corrosion datasets, respectively. Up to 84% filters being pruned, the mean detection accuracy after fine-tuning for crack and corrosion is approximately 99% with a standard deviation below 0.5%. This demonstrates the robustness of the proposed approach as the variations in the performance are extremely small when the pruned network still has the capacity to deal with the detection task. When 97% of the filters



(a)



(b)

FIGURE 15 Mean detection accuracy from fivefold cross-validation of pruned VGG16 versus percentage of pruned filters: (a) crack dataset and (b) corrosion dataset

TABLE 2 Mean (μ) and standard deviation (σ) of detection accuracy before and after network fine-tuning—VGG16 with crack datasets

Pruned filters (%)	μ (%) Before	μ (%) After	σ (%) Before	σ (%) After
0	99.9	99.9	0.11	0.11
12	91.1	99.7	2.43	0.12
24	98.8	99.7	1.33	0.11
36	95.6	99.8	4.05	0.06
48	93.0	99.9	8.95	0.05
60	82.5	99.7	9.12	0.09
72	73.3	99.8	19.67	0.10
84	53.8	99.4	4.59	0.19
97	50.3	84.7	0.52	19.86

TABLE 3 Mean (μ) and standard deviation (σ) of detection accuracy before and after network fine-tuning—VGG16 with corrosion datasets

Pruned filters (%)	μ (%) Before	μ (%) After	σ (%) Before	σ (%) After
0	99.7	99.7	0.28	0.28
12	95.3	99.1	1.76	0.12
24	98.4	99.3	0.27	0.21
36	97.1	99.5	1.40	0.11
48	95.6	99.4	1.20	0.05
60	87.2	99.3	6.26	0.13
72	77.0	99.0	16.98	0.20
84	65.0	98.4	10.20	0.31
97	49.5	96.0	0.74	0.95

are removed (i.e., only 128 filters left in VGG16), the mean accuracy of crack detection drops to 84.7% with a standard deviation 19.86%, and the mean accuracy of corrosion detection drops to 96.0% with a standard deviation 0.95%. This indicates that the pruning should terminate due to the increasing variation and decreasing accuracy in the detection performance. For the accuracy before fine-tuning, the mean exhibits a monotonic decreasing behavior as the pruning proceeds. The standard deviation before fine-tuning is quite high, which is due to the quality of random optimization for the remaining parameters in the network. For instance, if the values of the remaining filters are not fine-tuned very well, the accuracy before fine-tuning may drop more significantly, compared to the situation in which the remaining parameters are fine-tuned better. This explains why the accuracy before fine-tuning, as shown in Figure 6, exhibits an oscillation behavior. Notice that the variation in the accuracy before fine-tuning is very small when 97% of filters are removed, since the mean accuracy is 50%, meaning that the network performance is equal to random guesses. Lastly, while conventional transfer learning usually fine-tunes only the FC layers when there is no sufficient training data, the authors observe that fine-tuning the whole network leads to better detection performance for new tasks very different from the original task. It is noted that cross-validation is not necessary for deep learning if there exists sufficient available data. However, as transfer learning-based approaches usually use a small amount of data, cross-validation may enhance the reliability in the capability of generalization.

4.3 | Optimization for VGG16 feature extraction

To further reduce the inference time, the optimization approach described in Section 3.2 is implemented to eliminate the repetitive computation due to the overlapping regions between adjacent sliding windows. For demonstration

purposes, the pruned VGG16 network where 84% of kernels are removed is selected for crack detection, and the server setup is used as the computing platform. Since the pruned network is trained using 224×224 image patches as input, it is necessary to retrain the network by removing zero-paddings in the convolution layers for consistency. The reason is that the optimized approach takes the whole image as input, and the patches in the image must be no zero-padding. As shown in Section 4.2, it takes 4.04 s to process 3,720 patches for the pruned VGG16 network when 3,720 patches are passed to the network separately. By using the whole image as input and extracting features based on the proposed optimization approach, it takes only 0.62 s, which is 6.5 times faster than the approach without optimization.

5 | CONCLUSIONS

This study presents an approach for developing efficient DCNNs for damage detection in SHM. This is critical since an effective inspection system requires rapid inference and onboard deployment on mobile inspection devices, especially for large-scale civil infrastructures. The results in Section 4.1 show that the conventional transfer learning-based approach alone is not adequate for rapid inference as the resulting network spends the majority of time on computing the features. Section 4.2 demonstrates that by network pruning, the inference time of the pruned VGG16 network is nine times faster than the original network, and the memory demand is reduced by 80% without trading the detection performance. To reduce the pruning time spent on the server machine, a sensitivity analysis is conducted. The analysis indicates that using a smaller number of fine-tuning epochs during pruning reduces the pruning computation time and does not lead to a significant drop in detection accuracy. In addition, pruning ResNet18 reduces the inference time on the edge device to 8.9 s, which is 3.55 times faster than VGG16. Results from five-fold cross-validation demonstrate the robustness of the proposed approach, as the mean detection accuracy is 99% with a standard deviation below 0.5%. Network pruning should stop when 97% of the filters are removed from VGG16. Finally, Section 4.3 shows that the inference time is reduced by a factor of 6.5 through the proposed optimization approach for feature computing.

As data availability and resource efficiency are critical issues for field applications, when designing a DCNN to solve a specific domain problem, one should consider whether it is necessary to construct a large network. By exploiting transfer learning and network pruning, one can construct DCNNs without the need of a huge amount of training data. The efficiency in memory requirement and inference time is achieved without losing performance in damage detection through the reduction of the network size. It is worth mentioning that in some applications, such as nuclear power plant domes, structural and nonstructural cracks are both critical since the nonstructural cracks can lead to deterioration of underneath surfaces due to infiltration of water and air. For some other applications, however, such as earthquake reconnaissance where the nonstructural cracks are not important, one needs to develop approaches to differentiate between structural and nonstructural cracks.

ACKNOWLEDGMENTS

This research was partially sponsored by the National Science Foundation (NSF) Award IIS-1636891 "BD Spokes: Planning: MIDWEST: Cyberinfrastructure to Enhance Data Quality and Support Reproducible Results in Sensor Originated Big Data" and by the U.S. Army Research Laboratory and the U.K. Ministry of Defence under Agreement Number W911NF-16-3-0001. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Army Research Laboratory, the U.S. Government, the U.K. Ministry of Defence, or the U.K. Government. The U.S. and U.K. Governments are authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation hereon.

REFERENCES

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., … Zheng, X. (2016). TensorFlow: A system for large-scale machine learning. In *12th USENIX symposium on Operating Systems Design and Implementation (OSDI)* (Vol. 16, pp. 265–283). New York: ACM Press.

Aghasi, A., Abdi, A., Nguyen, N., & Romberg, J. (2017). Net-trim: Convex pruning of deep neural networks with performance guarantee. *arXiv preprint arXiv:1611.05162v4*.

Akbar, M. A., Qidwai, U., & Jahanshahi, M. R. (2019). An evaluation of image based structural health monitoring using integrated UAV platform. *Structural Control and Health Monitoring*, *26*(1). https://doi.org/10.1002/stc.2276.

Alvarez, J. M., & Salzmann, M. (2016). Learning the number of neurons in deep networks. *Advances in Neural Information Processing Systems*, *29*, 2262–2270.

Amezquita-Sanchez, J., & Adeli, H. (2015). Synchrosqueezed wavelet transform-fractality model for locating, detecting, and quantifying damage in smart highrise building structures. *Smart Materials and Structures*, *24*, 065034.

ASCE (American Society of Civil Engineers). (2017). 2017 Report card for America's infrastructure.

Atha, D. J., & Jahanshahi, M. R. (2018). Evaluation of deep learning approaches based on convolutional neural networks for corrosion detection. *Structural Health Monitoring*, *17*, 1110–1128. https://doi.org/10.1177/1475921717737051

Bertino, E., & Jahanshahi, M. R. (2018). Adaptive and cost-effective collection of high-quality data for critical infrastructure and emergency management in smart cities - Framework and challenges. *ACM Journal of Data and Information Quality (JDIQ)*, *10*(1). https://doi.org/10.1145/3190579.

Bottou, L. (2010). Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010* (pp. 177–186).

Brambilla, M., Ferrante, E., Birattari, M., & Dorigo, M. (2013). Swarm robotics: A review from the swarm engineering perspective. *Swarm Intelligence*, *7*(1), 1–41.

Cao, Q. D., & Choe, Y. (2018). Deep learning based damage detection on post-hurricane satellite imagery. *arXiv preprint arXiv:1807.01688v1*.

Cha, Y., Choi, W., Suh, G., Mahmoudkhani, S., & Büyüköztürk, O. (2018). Autonomous structural visual inspection using region-based deep learning for detecting multiple damage types. *Computer-Aided Civil and Infrastructure Engineering*, *33*(9), 731–747.

Cha, Y.-J., Choi, W., & Büyüköztürk, O. (2017). Deep learning-based crack damage detection using convolutional neural networks. *Computer-Aided Civil and Infrastructure Engineering*, *32*, 361–378.

Chang, C.-C., & Lin, C.-J. (2011). LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, *2*, 27:1–27:27. Software available at http://www.csie.ntu.edu.tw/cjlin/libsvm

Chen, F. C., & Jahanshahi, M. R. (2018). NB-CNN: Deep learning-based crack detection using convolutional neural network and naïve Bayes data fusion. *IEEE Transactions on Industrial Electronics*, *65*, 4392–4400.

Chen, F. C., Jahanshahi, M. R., Wu, R. T., & Joffe, C. (2017). A texture-based video processing methodology using Bayesian data fusion for autonomous crack detection on metallic surfaces. *Computer-Aided Civil and Infrastructure Engineering*, *32*, 271–287.

Choi, J., Yeum, C. M., Dyke, S. J., & Jahanshahi, M. R. (2018). Computer-aided approach for rapid post-event visual evaluation of a building facade. *Sensors*, *18*(9). https://doi.org/10.3390/s18093017

Chollet, F. (2015). keras. *GitHub*. https://github.com/fchollet/keras.

Dalal, N., & Triggs, B. (2005). Histograms of oriented gradients for human detection. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)* (Vol. 1, pp. 886–893).

Fan, R.-E., Chang, K.-W., Hsieh, C.-J., Wang, X.-R., & Lin., C.-J. (2008). LIBLINEAR: A library for large linear classification. *Journal of Machine Learning Research*, *9*, 1871–1874.

Gao, Y., and Mosalam, K. (2018). Deep transfer learning for image-based structural damage recognition. *Computer-Aided Civil and Infrastructure Engineering*, *33*(9), 748–768.

Gopalakrishnan, K., Gholami, H., Vidyadharan, A., Choudhary, A., & Agrawal, A. (2018). Crack damage detection in unmanned aerial vehicle images of civil infrastructure using pre-trained deep learning model. *International Journal for Traffic and Transport Engineering*, *8*(1), 1–14.

Guo, Y., Yao, A., & Chen, Y. (2016). Dynamic network surgery for efficient DNNS. *arXiv preprint arXiv:1608.04493v2*.

Han, S., Liu, X., Mao, H., Pu, J., Pedram, A., Horowitz, M. A., & Dally, W. J. (2016). EIE: Efficient inference engine on compressed deep neural network. In *Proceedings of the 43rd International Symposium on Computer Architecture* (pp. 243–254). Piscataway, NJ: IEEE Press.

Han, S., Mao, H., & Dally, W. J. (2016). Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding. *arXiv preprint arXiv:1510.00149v5*.

Han, S., Pool, J., Tran, J., & Dally, W. J. (2015). Learning both weights and connections for efficient neural networks. *arXiv preprint arXiv:1506.02626v3*.

He, K., Zhang, X., Ren, S., & Sun, J. (2015a). Deep residual learning for image recognition. *arXiv preprint arXiv:1512.03385*.

He, K., Zhang, X., Ren, S., & Sun, J. (2015b). Spatial pyramid pooling in deep convolutional networks for visual recognition. *arXiv preprint arXiv:1406.4729v4*.

He, Y., Lin, J., Liu, Z., Wang, H., Li, L.-J., & Han, S. (2019). AMC: AutoML for model compression and acceleration on mobile devices. *arXiv preprint arXiv:1802.03494v4*.

Hu, H., Peng, R., Tai, Y.-W., & Tang, C.-K. (2016). EIE: Efficient inference engine on compressed deep neural network. *arXiv preprint arXiv:1607.03250*.

Hu, H., Peng, R., Tai, Y.-W., & Tang, C.-K. (2018). Network trimming: A data-driven neuron pruning approach towards efficient deep architectures. *arXiv preprint arXiv:1607.03250v1*.

Kong, X., & Li, J. (2018). Vision-based fatigue crack detection of steel structures using video feature tracking. *Computer-Aided Civil and Infrastructure Engineering*, *33*(9), 783–799.

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems (NIPS)* (pp. 1106–1114). Cambridge, MA: MIT Press.

Kumar, S. S., Abraham, D. M., Jahanshahi, M. R., Iseley, T., & Starr, J. (2018). Automated defect classification in sewer closed circuit television inspections using deep convolutional neural networks. *Automation in Construction*, *91*, 273–283.

Lebedev, V., & Lempitsky, V. (2016). Fast ConvNets using group-wise brain damage. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (pp. 2554–2564).

LeCun, Y., Denker, J. S., Solla, S., Howard, R. E., & Jackel, L. D. (1990). Optimal brain damage. In *Advances in Neural Information Processing Systems (NIPS)*. Cambridge, MA: MIT Press.

Li, Z., Park, H., & Adeli, H. (2017). New method for modal identification and health monitoring of superhighrise building structures using discretized synchrosqueezed wavelet and Hilbert transforms. *Structural Design of Tall and Special Buildings*, *26*(3). https://doi.org/10.1002/tal.1312

Lin, Y. Z., Nie, Z. H., & Ma, H. W. (2017). Structural damage detection with automatic feature-extraction through deep learning. *Computer-Aided Civil and Infrastructure Engineering*, *32*(12), 1025–1046.

Liu, Z., Sun, M., Zhou, T., Huang, G., & Darrell, T. (2018). Rethinking the value of network pruning. *arXiv preprint arXiv:1810.05270v1*.

Luo, J.-H., Wu, J., & Lin, W. (2017). ThiNet: A filter level pruning method for deep neural network compression. *arXiv preprint arXiv:1707.06342v1*.

Maeda, H., Sekimoto, Y., Seto, T., Kashiyama, T., & Omata, H. (2018). Road damage detection using deep neural networks with

images captured through a smartphone. *arXiv preprint arXiv: 1801.09454v2*.

Mallya, A., & Lazebnik, S. (2018). PackNet: Adding multiple tasks to a single network by iterative pruning. *arXiv preprint arXiv:1711.05769v2*.

Molchanov, P., Tyree, S., Karras, T., Aila, T., & Kautz, J. (2017). Pruning convolutional neural networks for resource efficient inference. *arXiv preprint arXiv:1611.06440v2*.

NVIDIA Jetson Solutions for Drones and UAVs (2016). https://www.nvidia.com/en-us/autonomous-machines/uavs-drones-technology/

Oh, B. K., Kim, K., Kim, Y., Park, H. S., & Adeli, H. (2017). Evolutionary learning based sustainable strain sensing model for structural health monitoring of high-rise buildings. *Applied Soft Computing*, *58*, 576–585.

Ojala, T., Pietikainen, M., & Maenpaa, T. (2002). Multiresolution gray-scale and rotation invariant texture classification with local binary patterns. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *24*(7), 971–987.

Park, K., Torbol, M., & Kim, S. (2018). Vision-based natural frequency identification using laser speckle imaging and parallel computing. *Computer-Aided Civil and Infrastructure Engineering*, *33*(1), 51–63.

Park, S. W., Park, H. S., Kim, J., & Adeli, H. (2015). 3d displacement measurement model for health monitoring of structures using a motion capture system. *Measurement*, *59*, 352–362.

Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., … Lerer, A. (2017). Automatic differentiation in PyTorch. In *NIPS-W*. Cambridge, MA: MIT Press.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., … Duchesnay, É. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, *12*, 2825–2830.

Qarib, H., & Adeli, H. (2014). Recent advances in health monitoring of civil structures. *Scientia Iranica - Transaction A: Civil Engineering*, *21*(6), 1733–1742.

Rafiei, M. H., & Adeli, H. (2017). A novel machine learning based algorithm to detect damage in highrise building structures. *Structural Design of Tall and Special Buildings*, *26*(18). https://doi.org/10.1002/tal.1400

Rafiei, M. H., & Adeli, H. (2018). A novel unsupervised deep learning model for global and local health condition assessment of structures. *Engineering Structures*, *156*(1), 598–607.

Rafiei, M. H., Khushefati, W., Demirboga, R., & Adeli, H. (2017). Supervised deep restricted Boltzmann machine for estimation of concrete compressive strength. *ACI Materials Journal*, *114*(2), 237–244.

Shi, W., Cao, J., Zhang, Q., Li, Y., & Xu, L. (2016). Edge computing: Vision and challenges. *IEEE Internet of Things Journal*, *3*(5), 637–646.

Simonyan, K., & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.

Tang, J., Sun, D., Liu, S., & Gaudiot, J. L. (2017). Enabling deep learning on IoT devices. *Computer*, *50*(10), 92–96.

Verma, D. C., & Mel, G. D. (2017). Measures of network centricity for edge deployment of IOT applications. In *2017 IEEE International Conference on Big Data (Big Data)*. https://doi.org/10.1109/BigData.2017.8258505

Wu, R. T., & Jahanshahi, M. R. (2018a). Data fusion approaches for structural health monitoring and system identification: Past, present and future. *Structural Health Monitoring*. https://doi.org/10.1177/1475921718798769

Wu, R. T., & Jahanshahi, M. R. (2018b). Deep convolutional neural network for structural dynamic response estimation and system identification. *Journal of Engineering Mechanics (ASCE)*, *145*(1). https://doi.org/10.1061/(ASCE)EM.1943-7889.0001556

Xue, Y. D., & Li, Y. (2018). A fast detection method via region-based fully convolutional neural networks for shield tunnel lining defects. *Computer-Aided Civil and Infrastructure Engineering*, *33*(8), 638–654.

Yeum, C. M., Dyke, S. J., & Ramirez, J. (2018). Visual data classification in post-event building reconnaissance. *Engineering Structures*, *155*, 16–24.

Yeum, C. M., Dyke, S., Ramirez, J., & Benes, B. (2016). Big visual data analytics for damage classification in civil engineering. In *Proceedings of the International Conference on Smart Infrastructure and Construction*.

Yu, J., Lukefahr, A., Palframan, D., Dasika, G., Das, R., & Mahlke, S. (2017). Scalpel: Customizing DNN pruning to the underlying hardware parallelism. In *ISCA '17: Proceedings of the 44th Annual International Symposium on Computer Architecture* (pp. 548–560). New York, NY: ACM.

Zhang, A., Wang, K. C. P., Li, B., Yang, E., Dai, X., Peng, Y., … Chen, C. (2017). Automated pixel-level pavement crack detection on 3d asphalt surfaces using a deep-learning network. *Computer-Aided Civil and Infrastructure Engineering*, *32*(10), 805–819.

Zhou, A., Yao, A., Guo, Y., Xu, L., & Chen, Y. (2017). Incremental network quantization: Towards lossless CNNs with low-precision weights. *arXiv preprint arXiv:1702.03044v2*.

Zhou, H., Alvarez, J. M., & Porikli, F. (2016). Less is more: Towards compact CNNs. In *European Conference on Computer Vision* (pp. 662–677).

How to cite this article: Wu R-T, Singla A, Jahanshahi MR, Bertino E, Ko BJ, Verma D. Pruning deep convolutional neural networks for efficient edge computing in condition assessment of infrastructures. *Comput Aided Civ Inf*. 2019;34:774–789. https://doi.org/10.1111/mice.12449